

## APPENDIX A

### Sample HTML Source For An SS-DHTML Document Object

```
<html>
<head>
<title></title>
</head>

<body bgcolor="#C0C0C0">
<script Language="JavaScript1.2">
<!--
    function signForm() {
        var alltext = new String(document.po.T1.value +
document.po.T2.value + document.po.T3.value + document.po.S1.value +
document.po.T4.value);
        var ret;
        ret = document.Invisible.processInput(document.forms[0],
alltext);
        alert (ret);
    }
//-->
</script>
<em><big><font face="Arial Black">

<p>Form #1176 Purchase Order Request:</font></big></em>

    <APPLET Name = "Invisible" CODE = "InvisibleApplet.class" ARCHIVE =
"invisible.jar" WIDTH = 350 HEIGHT = 350 >
    </APPLET>

</p>
<a href="http://st9:8080/SWBCDemos/Forms/formsOnline.htm"><font
face="Garamond"><strong>

<p align="center">I want To Return To The Main Forms Page
</strong></font></a></p>

<hr size="3" width="50%" align="center">

<form name="po">
    <b><strong><big><big><p>Employee Name:</big></big></strong>
</b><input type="text"
name="T1" size="30"> <br>
    <br>
    <b><strong><big><big>Product Name:</big></big></strong> </b><input
type="text" name="T2"
size="35"> <br>
    <br>
    <b><strong><big><big>Reasons For Order:</big></big></strong> </b><br>
```



## Sample Java™ Signature Processing Applet For An SS-DHTML Document Object

```

/*****
/* file - InvisibleApplet.java
/* -----
/* This is the class definition for the applet that works through
/* LiveConnect with the forms in the Form Signature demonstration
/* This applet holds two responsibilities:
/* 1 - Authenticate the user and obtain their private key
/* 2 - Accept through a call from a JavaScript a reference to a form
/*      object and user specific data and sign this concatenated data
/*      with the private key obtained from authentication
/* -----
/* Revision Timeline:
/* 6/5 - 6/9 Attempted implementation of iButtons to perform authentic
/* 6/10 Decided a software implementation was a better starting point
/* 7/14 Moved search for user's private key to method that signs data *
/*****
/
import com.ms.security.*;
import java.security.*;
import java.io.*;
import java.util.*;
import java.net.*;
import com.oreilly.servlet.*;
import javax.crypto.*;
import javax.crypto.spec.*;
import javax.crypto.spec.PBEKeySpec;
import au.net.aba.crypto.provider.ABAProvider;
import au.net.aba.crypto.*;
import au.net.aba.security.spec.*;

public class InvisibleApplet extends java.applet.Applet {

    Instruct s = new Instruct();
    Message resul = new Message();
    SymAction lSymAction = null;
    Password p = new Password();
    PrivateKey session_key = null;
    PublicKey verify_key = null;
    byte[] sig = null;
    int fails = 0;
    boolean flag = false;
    BASE64Encoder coder = new BASE64Encoder();
    BASE64Decoder decoder = new BASE64Decoder();
    String encodedPubKey = "";

    byte[] salt = new byte[8];
    int iterations = 20;

/*****

```

```

/* The initialization method of the applet. Here there*/
/* are two main purposes. 1: Authenticate the user */
/* this will be done by what ever method chosen and 2:*/
/* Obtain the users private key off the floppy drive. */
/*****/
public void init() {

    try {

        System.out.println("-----");
        System.out.println("Secure Web Based Computing Test Bed");
        System.out.println("\nOnline Form System Demonstration");
        System.out.println("Version 4.00");
        System.out.println("-----\n");

        // Written solely for the Win32 VM. Since this is the init()
        // method we need to assert our specified permission to the
        // call stack in order to display a window giving
instructions
        if (Class.forName("com.ms.security.PolicyEngine") != null) {
            PolicyEngine.assertPermission(PermissionID.SYSTEM);
        }

        Security.addProvider(new ABAProvider());

        resul.setSize(400,200);

        try {

            encodedPubKey = getParameter("publicKey");

            if (encodedPubKey == null) {
                s.setVisible(true);
            }

        } catch (Exception e) {

            // Set the instructions dialog box visible
            s.setVisible(true);
            System.out.println(e);

        }

        // Declare an instance of the ActionHandling class
        // And attach listeners to each of the buttons
        lSymAction = new SymAction();
        p.button1.addActionListener(lSymAction);
        s.button1.addActionListener(lSymAction);
        resul.button1.addActionListener(lSymAction);

    } catch (Exception e) {
        System.out.println("Caught Exception in Applet Init(): " + e);
    }

} /* End init() */

/*****/

```

00736904 2400  
00736904 2400

```

/* This method added 7/14 to run the verification and */
/* its parameters to be obtained from the script that will */
/* call it. The servlet the hands this page to the client */
/* will have encoded the public key in the page as a */
/* parameter to obtain */
/* ***** */
public boolean processVerify(String form, String client_input,
String signature) {
    byte[] temp      = null;
    byte[] temp1     = null;
    byte[] sig       = null;
    byte[] ser_pubKey = null;
    byte[] fin       = null;
    boolean result    = false;

    try {
        System.out.println(signature);

        try {
            sig = decoder.decodeBuffer(signature);
        } catch (Exception e) {
            e.printStackTrace();
        }

        System.out.println("Client input in verify: " + client_input);

        ser_pubKey = decoder.decodeBuffer(encodedPubKey);
        ByteArrayInputStream bais = new ByteArrayInputStream(ser_pubKey);
        ObjectInputStream ois = new ObjectInputStream(bais);
        verify_key = (PublicKey) ois.readObject();
        ByteArrayOutputStream baos1 = new ByteArrayOutputStream();
        PrintWriter pw = new PrintWriter(baos1);
        pw.flush();
        pw.println(form);
        pw.flush();
        pw.close();
        temp1 = baos1.toByteArray();
        BASE64Encoder be = new BASE64Encoder();
        System.out.println("In Verify rep of form: " + be.encode(temp1));

        String concat = client_input;

        ByteArrayOutputStream baos = new ByteArrayOutputStream();
        ObjectOutputStream oos = new ObjectOutputStream(new
        BufferedOutputStream(baos));
        oos.writeObject(concat);
        oos.close();
        temp = baos.toByteArray();

        fin = new byte[temp.length + temp1.length];

        System.arraycopy(temp, 0, fin, 0, temp.length);
        System.arraycopy(temp1, 0, fin, temp.length, temp1.length);

        // Declare an instance of the DSA signature class from the JCA
        Signature dsa = Signature.getInstance("DSA");
        dsa.initVerify(verify_key);

```

```

dsa.update(fin);
result = dsa.verify(sig);

System.out.println(result);

} catch (Exception e) {
    System.out.println(e);
}

return result;

} /* end processVerify() */

/*****
/* Here is the workhorse method of the applet: The params */
/* to this applet are the reference to the form and the */
/* concatenated form input as one string. This information */
/* will also be concatenated and signed. */
*****/
public String processInput(String form, String client_input) {

    // Declare a byte array that will eventually hold the
    // signature on all the data supplied to the method
    byte[] tmp = null;
    byte[] tmp1 = null;
    byte[] fin = null;

    String return_value = "";

    // As an added check, determine if the private key
    // to user is null
    if (session_key == null) {
        System.out.println("There was a problem setting the private
key");
    } else {
        System.out.println("Before try statement");

        try {

            ByteArrayOutputStream baos1 = new ByteArrayOutputStream();
            PrintWriter pw = new PrintWriter(baos1);
            pw.flush();
            pw.println(form);
            pw.flush();
            pw.close();
            tmp1 = baos1.toByteArray();

            BASE64Encoder be = new BASE64Encoder();
            System.out.println("In Sign rep of form: " + be.encode(tmp1));

            String concat = client_input;
            ByteArrayOutputStream baos = new ByteArrayOutputStream();
            ObjectOutputStream oos = new ObjectOutputStream(new
BufferedOutputStream(baos));
            oos.writeObject(concat);
            oos.close();
            tmp = baos.toByteArray();

```

```

        fin = new byte[tmp.length + tmp1.length];

        System.arraycopy(tmp, 0, fin, 0, tmp.length);
        System.arraycopy(tmp1, 0, fin, tmp.length, tmp1.length);

        // Declare an instance of the DSA signature class from the JCA
        Signature dsa = Signature.getInstance("SHA/DSA");
        dsa.initSign(session_key);
        dsa.update(fin);

        // And sign the input from the client
        sig = dsa.sign();

        }catch (Exception e) {
            System.out.println(e);
        }

        // In case of error where signature was not created properly
        // examine the byte array
        if (sig != null) {

            // Here the signature is fine so return it to the calling
            JavaScript
                return_value = coder.encode(sig);

            }else {

                // Here the signature was not fine so to avoid a
                NullPointerException
                // assemble a random byte array and inform the user of the
                problem
                System.out.println("The objects were not properly stored
                together");
                tmp = new byte[1];
                return_value = tmp.toString();
            }

        }
        return return_value;
    } /* End processInput() */

    /*****
    /* Action Handling Class for AWT components */
    /*****/
    class SymAction implements java.awt.event.ActionListener {

        // decide which component was triggered and execute appropriate
        // handler for it
        public void actionPerformed(java.awt.event.ActionEvent event) {

            try {

                Object object = event.getSource();

```

```

        if (object == s.button1) {
            sButton1_ActionPerformed(event);
        }

        if (object == p.button1) {
            pButton1_ActionPerformed(event);
        }

        if (object == resul.button1) {
            resul.setVisible(false);
        }

    } catch (NullPointerException e) {
        System.out.println(e);
    }

} /* End actionPerformed() */

} /* End Inner class SymAction() */

/*****
/* Action Handling method for the Continue button on the */
/* Instructions form, class Instruct */
*****/
void sButton1_ActionPerformed(java.awt.event.ActionEvent event) {

    try {

        // User acknowledged inserting his private key disk into floppy
        // now he wants to continue, so minimize this dialog box
        s.setVisible(false);

        // To Check and see if there is a disk in the floppy drive
        // Form a file instance of A:\. Any NullPointerException will
        // indicate that the user has not inserted a disk
        File testFile = null;
        String floppy_test = ("A:" + File.separator);
        testFile = new File(floppy_test);

        if ((testFile.exists()) == false) {
            try {
                fails++;
                s.wrappingLabel1.setText("There is no disk in the
floppy drive, please try again");
                s.setVisible(true);
            } catch (Exception e2) {
                System.out.println(e2);
            }
        } else {

            if (fails < 3) {
                // And present the user with a chance to enter their password
                // by showing the password dialog box
                p.setVisible(true);
            } else {
                /* end inside else{} */
            }
        }
    }
}

```



```

    }

    }catch (NullPointerException e) {

        // Depending on which VM is used, a file instance that does not
        exist
        // can throw a NullPointerException, so catch for it here
        // Since the above test for the floppy failed, redisplay the
        // Instruct instance will the following instructions
        try {

            s.wrappingLabel1.setText("No disk could be located in the
            floppy drive, please try again.");
            s.setVisible(true);

        }catch (Exception e1) {
            System.out.println(e1);
        }

        }catch (Exception e) {
            // Catch all other exceptions
            System.out.println(e);
        }
    }

} /* End sButton1_ActionPerformed() */

/*****
/* Action Handling method for the Password Dialog Box      */
/* This is where the private key will be obtained from the  */
/* floppy drive                                             */
*****/
void pButton1_ActionPerformed(java.awt.event.ActionEvent event) {

    try {

        // Minimize the password dialog box
        p.setVisible(false);
        resul.setVisible(true);
        resul.textArea1.append("Searching For Employee Information
        File....");

        // Now is time to check if user inserted correct disk,
        // File where data is held is titled by username, if no file
        // Exists then a FileNotFoundException will be thrown
        String path = new
        String("A:" + File.separator + p.textField1.getText());
        System.out.println("Debug File Path" + path);
        FileInputStream infoHandle = new FileInputStream(path);

        // A valid file existed if execution reaches this point,
        begin the input
        // of the object containing the private key
        ObjectInputStream ois = new ObjectInputStream(infoHandle);
        resul.textArea1.append(" Done\n");

        x509 temp = null;

```

```

resul.textArea1.append("Decrypting Encoded Signature
Key...");

// An appropriately Named Files exists on the disk, try to
extract
// an Instance of x509 from it
try {
temp = (x509) ois.readObject();
}catch (Exception e1) {
e1.printStackTrace();
}

// Declare the Secret Key and factory used to generate it
KeySpec ks = new
PBEKeySpec((p.textField2.getText()).toCharArray());
SecretKeyFactory skf =
SecretKeyFactory.getInstance("PBEWithMD5AndDES");
SecretKey key = skf.generateSecret(ks);

// Declare the new salt to use in decryption
byte[] salt = temp.salt;

// Create an instance of the parameter spec to encapsulate all
// parameters to the cipher in one class
AlgorithmParameterSpec aps = new PBEParameterSpec(salt,
iterations);

// Create an instance of the Cipher that will decrypt the key
Cipher cipher = Cipher.getInstance("PBEWithMD5AndDES");

// This is a symmetric cipher so initialize it to decryption
mode
cipher.init(Cipher.DECRYPT_MODE, key, aps);

// And decrypt it
byte[] serialized_key =
cipher.doFinal(decoder.decodeBuffer(temp.passwd));
resul.textArea1.append(" Done\n");

resul.textArea1.append("Extracting Private Key From Decrypted
Data...");

// Extract the serialized object from the decrypted key
ByteArrayInputStream bais = new
ByteArrayInputStream(serialized_key);
ObjectInputStream ois2 = new ObjectInputStream(new
BufferedInputStream(bais));

// And make sure it is the DSA private key
session_key = (PrivateKey) ois2.readObject();
resul.textArea1.append(" Done\n");

resul.textArea1.append("You Many Now fill Out Your Form, to
submit press the submit button.");

```

```

}catch (FileNotFoundException e) {

    try {

        // The file does not exist on the disk that the client
        // Inserted so inform the client of this.
        fails++;
        p.setVisible(false);
        s.wrappingLabel1.setText("No such identification exists
on this disk, please try again");

        // Show the alert box
        s.setVisible(true);

    }catch (Exception e3) {
        System.out.println(e3);
    }

}catch (Exception e) {

    e.printStackTrace();

}/* End outermost try{}catch{} */

} /* End pButton1_ActionPerformed() */

} /* End InvisibleApplet() */

```

09735804-131400